**Introduction to CIM Cmdlets**

Rate This  
http://blogs.msdn.com/Utility/images/star-left-on.pnghttp://blogs.msdn.com/Utility/images/star-right-on.pnghttp://blogs.msdn.com/Utility/images/star-left-on.pnghttp://blogs.msdn.com/Utility/images/star-right-on.pnghttp://blogs.msdn.com/Utility/images/star-left-on.pnghttp://blogs.msdn.com/Utility/images/star-right-on.pnghttp://blogs.msdn.com/Utility/images/star-left-on.pnghttp://blogs.msdn.com/Utility/images/star-right-on.pnghttp://blogs.msdn.com/Utility/images/star-left-on.pnghttp://blogs.msdn.com/Utility/images/star-right-on.png

[http://i1.social.s-msft.com/profile/u/avatar.jpg?displayname=powershell+team&size=largePowerShell Team](http://blogs.msdn.com/338811/ProfileUrlRedirect.ashx)

PowerShell Team

Microsoft

MSFT

190,220 Points 7 4 5

Recent Achievements

First Answer Confirmed First Forums Reply New Gallery Storyteller

[View Profile](https://social.msdn.microsoft.com/profile/powershell%20team/?ws=usercard-hover)

24 Aug 2012 3:42 PM

* Comments [8](http://blogs.msdn.com/b/powershell/archive/2012/08/24/introduction-to-cim-cmdlets.aspx#comments)

PowerShell 3.0 shipping with Windows server 2012 and Windows 8 brings a new set of Cmdlets to manage any server or device that complies with CIM and WS-Man standards defined by DMTF. In this blog post we will explore these new Cmdlets and how can they help IT Pros in managing a datacenter.

The list of new Cmdlets is given in the table below:

|  |  |
| --- | --- |
| **Cmdlet** | **Purpose** |
| Get-CimInstance | Gets instances of a class. |
| New-CimInstance | Creates a new instance of a class. |
| Remove-CimInstance | Removes one of more instances of a class. |
| Set-CimInstance | Modifies one or more instances of a class. |
| Get-CimAssociatedInstance | Gets all the associated instances for a particular instance. |
| Invoke-CimMethod | Invokes instance or static method of a class. |
| Get-CimClass | Gets class schema of a CIM class. |
| Register-CimIndicationEvent | Helps subscribe to events. |
| New-CimSession | Creates a CIM Session with local or a remote machine |
| Get-CimSession | Gets a list of CIM Sessions that have been made. |
| Remove-CimSession | Removes CimSessions that are there on a machine. |
| New-CimSessionOption | Creates a set of options that can be used while creating a CIM session. |

**Basic terminology**

If you are already familiar with terms like WMI, CIM, WinRM and WS-Man, you can skip this section.

**CIM**: Common Information Model (CIM) is the DMTF standard [DSP0004] for describing the structure and behavior of managed resources such as storage, network, or software components.

**WMI**: Windows Management Instrumentation (WMI) is a CIM server that implements the CIM standard on Windows.

**WS-Man**: WS-Management (WS-Man) protocol is a SOAP-based, firewall-friendly protocol for management clients to communicate with CIM servers.

**WinRM:** Windows Remote Management (WinRM) is the Microsoft implementation of the WS-Man protocol on Windows.

**Why new Cmdlets**

With Windows Server 2012, Windows has shifted its focus to become a Cloud OS. In a datacenter running a private or public cloud – there are always set of devices and servers from different vendors. We strongly believe that the only way to reduce the cost of running datacenters is through automation built on top of standards. There is less learning and less chances of error when IT Pros are able to automate heterogeneous environment using same set of tools that they have learned for Windows.

This was not the case with previous versions of PowerShell. We had two set of Cmdlets to manage Windows and Non-Windows. WMI Cmdlets were primarily used to manage Windows and WsMan Cmdlets were targeted at non-Windows that implemented WsMan standard.

**WMI Cmdlets:**

- Pros: Provided better task abstraction compared to WsMan Cmdlets, output is a .NET object.

- Cons: Use of non-standard DCOM protocol. Does not work for non-Windows.

**WinRM Cmdlets**

- Pros: Works with Windows and non-Windows using standard protocol.

- Cons: Poor task abstraction. Output is XML , scaring most IT Pros away

We heard from you that the major obstacle in PS scripting for WMI is lack of discoverability. We also heard that WMI Cmdlets don’t provide first class PS experience. There are issues with serializing a WMI object, there is no concept of session reuse and WMI object has weird looking property names (like \_\_Server). Last but not least, there is poor formatting for most of the commonly used WMI classes. Someone said “When I run Get-WmiObject Win32\_process, I feel like standing on bridge above freeway and lot of traffic zipping in front of my eyes”.

**Key goals for new CIM Cmdlets**

- **Rich PowerShell experience**. Make CIM a first class citizen of PS, addressing usability concerns and user feedback for WMI and WsMan Cmdlets.

- **Standard compliance**. With so much focus on standards, our goal is to make PowerShell the best platform for managing Windows and Non-Windows. New CIM Cmdlets should be able to manage any CIM + WsMan compliant endpoint, including Windows.

- **Support for down-level machines**. We understand that there are more down-level servers in a datacenter than there would be Windows Server 2012 for some time to come. We want to make sure same set of Cmdlets can be used to manage down-level Windows as well.

In the subsequent sections we will talk about each of these goals in detail.

**Goal 1 - Rich PowerShell experience**

**1. Discovery of classes and namespaces.**

There is a famous joke about WMI, often attributed to Jeffrey Snover. “IT Pros love and hate WMI. They love it because there is so much they accomplish with WMI and PowerShell. They hate it because it is very difficult to discover what’s in there”

To address this, we have done key improvements in new CIM Cmdlets

1. Tab completion for classname and namespace parameters.
2. Get-CimClass Cmdlet

|  |
| --- |
| # Using tab completion for CIM cmdlet parameters ( Tab+Space in ISE shows a drop down)  Get-CimInstance –Namespace <Tab> #Finding top-level namespaces  # Tab completion for class names  # If namespace is not specified, shows classes from default root/cimv2 namespace  Get-CimInstance -ClassName \*Bios<Tab>  Get-CimInstance –Namespace root/Microsoft/Windows/smb –ClassName <tab>  # Note: Tab completion only works for local machine.  #Using Get-CimClass for advanced class search  #All classes in root/cimv2  PS:>Get-CimClass  #Classes named like disk  PS:>Get-CimClass -ClassName \*disk\*  # The Cmdlet makes querying much easier (what would require scripting before)  # Get all classes starting with "Win32" that have method starting with "Term"  PS:>Get-CimClass Win32\* -MethodName Term\*  # Get classes starting with "Win32" that have a property named "Handle"  PS:>Get-CimClass Win32\* -PropertyName Handle  # Get classes starting with "Win32" that have the "Association" qualifier  PS:>Get-CimClass Win32\* -QualifierName Association  #Find classes used for events  PS:> Get-CimClass -Namespace root/Microsoft/Windows/smb -class \*Smb\* -QualifierName Indication |

Get-CimClass and tab completion described above only works for local machine. If you specify ComputerName or CimSession parameter, tab completion would not work. In order to make tab completion more efficient, PowerShell caches the list of namespaces and classnames.

**2. Getting Instances**

Getting/Enumerating instance of a class is the most commonly performed operation. We wanted the new Get-CimInstance Cmdlet to have better performance and even better user experience as compared to the old Get-WmiObject cmdlet. To achieve these goals we made the following design decisions:

a. The Get-CimInstance cmdlet returns one or more instances of CimInstance. CimInstance is different from the object returned by Get-WmiObject.

b. \_\_Properties are no longer mixed with properties of an instance.

c. Reduce memory and on-the-wire footprint by allowing retrieval of a subset of properties.

d. Allow retrieval of key properties.

e. Allow creation of in-memory instance to reduce round trips.

f. Allow retrieval of specific instances using in-memory instances or actual instances.

g. DateTime values are returned as objects of System.DateTime type. Old cmdlets treat them as strings.

|  |
| --- |
| # Get-CimInstance was designed to be similar to the Get-WmiObject  # WMI Cmdlet : Get-WmiObject -class Win32\_Process  # WsMan Cmdlet : get-wsmaninstance wmicimv2/win32\_process -Enumerate  # The default value of -Namespace is root/cimv2, and the default value of -ComputerName is local computer  PS:> Get-CimInstance -Class Win32\_Process  # Filtering using WQL  PS:> Get-CimInstance -Query "SELECT \* FROM Win32\_Process WHERE Name Like 'power%'"  # use the -Filter parameter with -classname  PS:> Get-CimInstance -Class Win32\_Process -Filter "Name Like 'power%'"  #Retrieving a subset of properties : To reduce memory and on-the-wire footprint  PS:> Get-CimInstance -Class Win32\_Process -Property Name, Handle  #Only get the key properties  PS:> Get-CimInstance -Class Win32\_Process -KeyOnly  ########################## Looking into CimInstance #########################  PS:>  $x, $y = Get-CimInstance Win32\_Process  PS:>  $x | gm  # The object contains the full CIM class derivation hierarchy  PS:> $x.pstypenames  # The object also has a reference to its class declaration  PS:> $x.CimClass | gm  # DateTime values are returned as strings  PS:> Get-WmiObject Win32\_OperatingSystem | Select \*Time\*  # DateTime values are returned as System.DateTime  PS:> Get-CimInstance Win32\_OperatingSystem | Select \*Time\*# |

**3. Working with associations**

Working with associations was not straight forward in the old WMI cmdlets. Customers wanted a way to find out associated instances of a particular instance. To resolve this issue we came up with a cmdlet called Get-CimAssociatedInstance. The following example shows some of the important features of this cmdlet.

|  |
| --- |
| # Get instance of Win32\_LogicalDisk class with DriveType =3 (hard drives)  PS:> $disk1, $diskn = Get-CimInstance -class Win32\_LogicalDisk -Filter 'DriveType = 3'  PS:> $disk1  # Get the all instances associated with this disk  PS:> Get-CimAssociatedInstance -CimInstance $disk1  # Get instances of a specific type  PS:> Get-CimAssociatedInstance -CimInstance $disk1 -ResultClassName Win32\_DiskPartition  # Finding associated instances through a specific CIM relationship  PS:> Get-CimAssociatedInstance -CimInstance $diskn -Association Win32\_LogicalDiskRootDirectory |

**4. Working with methods**

A large number of tasks in WMI are achieved by invoking methods. The new Invoke-CimMethod makes two things really easy:

a. Discovery of method and method parameters

b. Execution of methods with parameters. Using feedback from our customers: [Connect bug](https://connect.microsoft.com/PowerShell/feedback/details/624263/invoking-wmi-methods-with-invoke-wmimethod)

The following examples show how easy it is to discover and execute methods using Invoke-CimMethod:

|  |
| --- |
| PS:> $class = Get-CimClass Win32\_Process  PS:> $class.CimClassMethods  # Get the parameters of the Create method  PS:> $class.CimClassMethods["Create"].Parameters  # Invoke the static Create method on the Win32\_Process class to create an instance of the Notepad  # application. Notice that the method parameters are given in a hash table since CIM method arguments  # are unordered by definition.  PS:> Invoke-CimMethod -Class win32\_process -MethodName Create -Argument @{CommandLine='notepad.exe';  CurrentDirectory = "c:\windows\system32"}  # Get the owners of the running Notepad instances  PS:> $result = Invoke-CimMethod -Query 'SELECT \* FROM Win32\_Process WHERE name like "notepad%"'  -MethodName GetOwner  # The result has the returned value and out parameters of the method  PS:> $result |

**5. CliXML serialization.**

CimInstance supports full fidelity serialization and deserialization. This is an important feature for those who wish to save state of an instance or result of a cmdlet and then want to use it later. The WMI cmdlets do not support full-fidelity serialization/deserialization.

|  |
| --- |
| # CimInstances are serialized and deserialized with full fidelity  PS:> $x = Get-CimInstance Win32\_Service  PS:> $x  PS:> $x[0].pstypenames  PS:> $x | Export-CliXml t1.xml  PS:> $y = Import-CliXml .\t1.xml  PS:> $y  PS:> $y[0].pstypenames  # The deserialized objects are identical to the ones obtained from the server  PS:> diff ($y) (Get-CimInstance win32\_service ) |

**6. Remote management**

Managing remote machines with the new cmdlets is also pretty simple and straight forward. The two parameters that can be used to manage remote machines are:

a. ComputerName

b. CimSession

The following example will explain how these two parameters can be used to manage a remote machine.

|  |
| --- |
| PS:> $props = @{v\_Key = [UInt64] 8;}  # If ComputerName parameter is used, the cmdlets create an implicit session during the execution.  PS:> $inst = New-CimInstance -ClassName TestClass -Namespace root\test -Key v\_Key -Property $props  -ComputerName SecondWin8Server  # Create a session  PS:> $session = New-CimSession –ComputerName SecondWin8Server  # Use the session  PS:> $inst = New-CimInstance -ClassName TestClass -Namespace root\test -Key v\_Key -Property $props  –CimSession $session |

**Tip:** If a large number of remote operations are to be performed then session reuse is highly recommended. This can provide a significant performance gain.

**7. Easy migration from WMI cmdlets**

Last, but not the least, a very important aspect is to reduce learning. If you know how to use WMI cmdlets from PowerShell 2.0, you will find learning new CIM cmdlets easy. These cmdlets follow similar patterns as WMI cmdlets -including parameter names and alias. Let’s see how easy or difficult it is to move to new cmdlets

|  |
| --- |
| **# OLD: One liner to get ComputerSystem information**  PS:> Get-WmiObject Win32\_ComputerSystem  **# NEW:**  PS:> Get-CimInstance Win32\_ComputerSystem  # ClassName is position and mandatory.  # Namespace default is root/cimv2 namespace  #WMI Cmdlet – with classnames and Namespace  PS:> Get-WmiObject –ClassName Win32\_Process –Namespace root/cimv2  #CIM cmdlet follows the same pattern  PS:> Get-CimInstance –ClassName Win32\_Process –Namespace root/cimv2 |

The table below shows the list of WMI cmdlets and their CIM equivalent:

|  |  |
| --- | --- |
| **Old cmdlet** | **New Cmdlet** |
| Get-WmiObject | Get-CimInstance |
| Get-WmiObject -list | Get-CimClass |
| Set-WmiInstance | Set-CimInstance |
| Set-WmiInstance –PutType CreateOnly | New-CimInstance |
| Remove-WmiObject | Remove-CimInstance |
| Invoke-WmiMethod | Invoke-CimMethod |

**Note:** In old cmdlets Set-WmiInstance was used to perform update and create operations. The new cmdlets clearly define the distinction between the two operations i.e. New-CimInstance is used to create an instance and Set-CimInstance is used to update an existing instance.

You will also notice that in some places we have broken this rule – and script using WMI cmdlet can’t be simply changed to CIM cmdlet by changing the cmdlet name. e.g.

|  |
| --- |
| **# OLD:**  PS:> Invoke-WMIMethod -class Win32\_Process -Name create -ArgumentList 'calc.exe'  **# NEW:**  PS:>Invoke-CimMethod Win32\_Process -MethodName create -Arguments @{CommandLine='calc.exe'}  New Cmdlet takes a hash table or ordered dictionary as input not an Object. |

The rule here is, if the WMI cmdlet experience is not ideal, we don’t want to carry it forward. It’s better to break a bad experience (like in the example given above)

**Goal 2 - Standard Compliance**

· CIM cmdlets are modeled on generic CIM operations

· Work over WsMan for remote management, making it possible to manage any server or device that implements WsMan and CIM standard.

· Removes the need to build custom agents and/or protocol to manage complex heterogeneous environments

· CIM cmdlets will work seamlessly with the new Open Management Infrastructure. <http://blogs.technet.com/b/windowsserver/archive/2012/06/28/open-management-infrastructure.aspx>

The following example shows that CimCmdlets can be used to manage a non-windows device (i.e. an Intel AMT box):

|  |
| --- |
| # Save credentials in an object  PS:> $cred = Get-Credential -UserName admin  # Create a session with Intel AMT machine.  PS:> $s = New-CimSession -ComputerName $serverAMT -Port 16992 -Authentication Digest -Credential $cred  # Use classname to get instances from an Intel AMT machine.  PS:> Get-CimInstance -class CIM\_ComputerSystem -Namespace interop -CimSession $s  # Use resourceURI to get instances from an Intel AMT machine.  PS:> $resourceURI = “http://intel.com/wbem/wscim/1/amt-schema/1/AMT\_GeneralSettings”  PS:> Get-CimInstance -ResourceUri $resourceURI -Namespace interop -CimSession $s |

**Goal 3 - Support for down-level OS or non-windows machines**

CimCmdlets can be used to manage Windows 8, down-level windows machines and any machine that has a CIM standards compliant CIMOM. So, IT pros can now manage non-windows machines with PowerShell in the exact same fashion as they could manage a windows box.

To manage these machines, a CimSession to the machine is required. Once a CimSession is created, the machine can be managed irrespective of the operating system.

WinRM support on down-level windows machines is limited. The latest WMF-V1 release will enable the latest version of WinRM on the down-level machines; this will help our customers in making best use of these new cmdlets.

If customers do not want to install WMF-V1 or do not wish to use WinRM then they can always use DCOM protocol to achieve the same results.

|  |
| --- |
| # If ComputerName is specified, WinRM protocol is used  PS:> $serverwin8 = New-CimSession –ComputerName "ServerWin8"  PS:> $serverwin2k8r2 = New-CimSession –ComputerName “Serverwin2k8r2 "  # For a remote machine, if you wish to go over DCOM, you need to explicitly specify  PS:> $sOpt = New-CimSessionOption –Protocol DCOM  PS:> $sessionWin2k8r2Dcom = New-CimSession –ComputerName “Serverwin2k8r2 " –SessionOption $sOpt  # User Experience remains same irrespective of the target machine.  PS:> $instanceServerWin8 = Get-CimInstance –CimSession $ serverwin8  PS:> $instanceServerWin2k8r2 = Get-CimInstance –CimSession $ serverwin2k8r2  PS:> $instanceServerWin2k8r2DCOM = Get-CimInstance –CimSession $ sessionWin2k8r2Dcom  #Note: The results in $instanceServerWin2k8r2 and $instanceServerWin2k8r2DCOM are identical in all aspects.  # Only the protocol used to get them is different. |

Our objective is to provide the best user experience to all the IT professionals. We would love to know what you think about these new cmdlets. Please give us your comments below.

Vaibhav Chugh

SDET, WMI

Osama Sajid

Program Manager, WMI

* [8 Comments](http://blogs.msdn.com/b/powershell/archive/2012/08/24/introduction-to-cim-cmdlets.aspx#comments)



|  |  |
| --- | --- |
|  |  |

[WMI](http://blogs.msdn.com/b/powershell/archive/tags/WMI/), [CIM](http://blogs.msdn.com/b/powershell/archive/tags/CIM/), [MI Client](http://blogs.msdn.com/b/powershell/archive/tags/MI+Client/), [CimInstance](http://blogs.msdn.com/b/powershell/archive/tags/CimInstance/)